5. CPU Registers and Instruction Cache

The TMS34010's on-chip CPU includes two general-purpose register files, file A and file B. Each register file contains 15 32-bit registers. The two files share a 32-bit hardware stack pointer (SP) that automatically manages the system stack during interrupts and subroutine calls. The CPU also contains two dedicated 32-bit registers – a program counter and a status register. An on-chip cache memory holds up to 128 instruction words, and is transparent to software. The CPU registers and instruction cache are discussed in the following sections:

In addition to the CPU registers, the TMS34010 contains 28 memory-mapped registers that are dedicated to I/O functions. These are described in Section 6.

5.1 General-Purpose Registers

The TMS34010 has 30 32-bit general-purpose registers, divided into register files A and B. In addition, a single stack pointer (SP) is common to both register files.

The multiple internal data paths linking the ALU and general-purpose registers provide single machine state execution of most register-to-register instructions. Single-state instructions include add, subtract, Boolean operations, and shifts (1 to 32 bits). During a single-state instruction, the following actions occur:

1) Two 32-bit operands are read in parallel from the general-purpose registers.

2) The specified operation is performed by the ALU.

3) The 32-bit result is stored in the specified general-purpose register.

The general-purpose registers are dual-ported to permit operands to be read from two independent registers at the same time.

5.1.1 Register File A

Fifteen of the 30 general-purpose registers, A0–A14, form register file A. These registers can be used for data storage and manipulation. No hardware-dedicated functions are associated with these general-purpose registers.

All register-to-register instructions (except MOVE RS,RD) require both registers to be in the same file. Instructions used to manipulate registers A0–A14 can also be used to manipulate the stack pointer. The SP can be specified in place of an A-file register in any of these instructions. Figure 5-1 illustrates register file A.

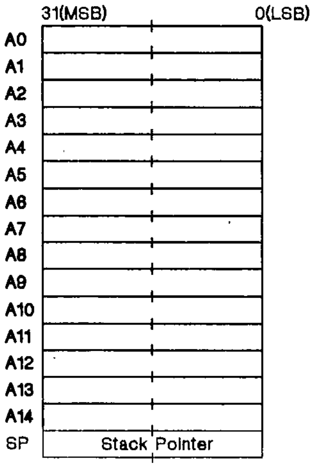


Figure 5-1. Register File A

5.1.2 Register File B

Register file B consists of 15 general-purpose registers, B0–B14. All register-to-register instructions (except MOVE RS,RD) require both registers to be in the same file. Instructions used to manipulate registers B0–B14 can also be used to manipulate the stack pointer. The SP can be specified in place of a B-file register in any of these instructions.

Registers B0–B14 can be used for general-purpose functions such as data storage and manipulation. During PixBlt and other pixel operations, however, these registers are assigned hardware-dedicated functions.

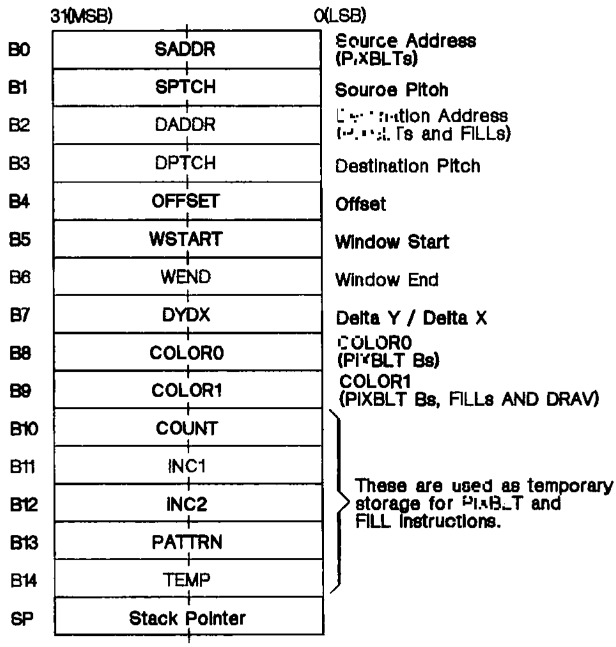


Figure 5-2. Register File B

As Figure 5-2 shows, registers B0–B9 are used as special-purpose registers during pixel operations. These registers must be loaded with specific parameters before execution of pixel operations. Registers B10–B14 are used as special-purpose registers for the LINE instruction. During pixel operations, registers B10–B14 are used for temporary storage; their previous contents are destroyed. Register functions may vary for individual instructions.

The B-file registers are described in detail in Section 5.1.4.

5.1.3 Stack Pointer

The stack pointer (SP), shown in Figure 5-3, is a 32-bit register that contains the bit address of the top of the system stack. Section 3.3 describes stack operation in detail. The SP appears as a member of both the A and B files, and can be specified as the operand in any instruction that manipulates the general-purpose registers. The machine contains only a single SP, but this SP can be addressed as a member of *either* register file, A or B.

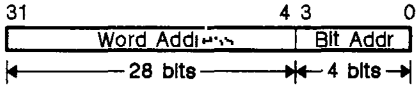


Figure 5-3. Stack Pointer Register

The system stack grows in the direction of smaller addresses. During an interrupt, the PC and ST are pushed onto the stack to permit the interrupted routine to resume execution when interrupt processing is completed. A subroutine call saves the PC on the stack to allow the calling routine to resume execution when subroutine execution is completed.

The stack pointer always points to the value at the top of the stack. Specifically, the SP contains the 32-bit address of the LSB of that value. While the four LSBs of the SP may be set to an arbitrary value, stack operations execute more efficiently when the four LSBs are 0s. Setting these bits to 0s aligns the stack pointer to 16-bit word boundaries in memory, reducing to two the number of memory cycles necessary to push or pop the contents of a 32-bit register.

The SP can be specified as the source or destination operand in any instruction that operates on the general-purpose registers. The SP can be accessed as register 15 in file A or B. Refer to the descriptions of the specific instructions for details.

5.1.4 Implied Graphics Operands

Table 5-1 summarizes the B-file register functions during pixel operations. These registers are referred to as *implied graphics operands*. Several I/O registers, described in Section 6, are also implied graphics operands. Individual descriptions of the B-file registers follow Table 5-1.

Table 5-1. B-File Registers Summary

Reg.	Function	Description
B0	SADDR	*Source Address.* Address of the upper left corner of the source pixel array (lowest pixel address in the array). SADDR is a linear or XY address, depending on the instruction which uses it.
B1	SPTCH	*Source Pitch.* Difference in linear start addresses between adjacent rows of a source pixel array.
B2	DADDR	*Destination Address.* Address of the upper left corner of the destination pixel array (lowest pixel address in the array). DADDR is a linear or XY address, depending on the instruction which uses it.
B3	DPTCH	*Destination Pitch.* Difference in linear start addresses between adjacent rows of a destination pixel array.
B4	OFFSET	*Offset.* Linear bit address corresponding to XY-coordinate origin (X=0, Y=0).
B5	WSTART	*Window Start Address.* XY address of the upper left corner of the window (smallest X and Y coordinate values in the array). 31–16: Starting Y 15–0: Starting X
B6	WEND	*Window End Address.* XY address of the lower right corner of the window (largest X and Y coordinate values in the array). 31–16: Ending Y 15–0: Ending X
B7	DYDX	*Delta Y/Delta X.* The 16 LSBs of this register specify the width (X dimension) of the source array in terms of either pixels or bits, depending on the instruction. The 16 MSBs specify the height (Y dimension) of the source array. If either DY = 0 or DX = 0 then nothing is moved. 31–16: Delta Y 15–0: Delta X

Table 5-1. B-File Registers Summary (Concluded)

Reg.	Function	Description
B8	Color 0	*Pixel value corresponding to "color 0".* COLOR0 contains the source background color to be used during a bit-expand operation (PIXBLT B,XY or PIXBLT B,L). The pixel value should be replicated throughout the 16 LSBs of register B8 (see note below). Non replicated patterns may be entered for dithering effects. The 16 MSBs are ignored during the expand operation. For example, at four bits per pixel, COLOR0 contains four identical pixel values, as shown below. 31 28 | 27 24 | 23 20 | 19 16 | 15 12 | 11 8 | 7 4 | 3 0 Pixel | Pixel | Pixel | Pixel | Pixel | Pixel | Pixel | Pixel Each of the 16 LSBs of COLOR0 is associated with the corresponding pin of the local address/data bus, LAD0–LAD15. COLOR0 bit 0 is associated with bit 0 of the data bus (the bit transferred on LAD0), COLOR0 bit 1 is associated with bit 1 of the data bus, and so on. When the contents of COLOR0 are output over a portion of the data bus, including a bit *n* of the bus, as an example, bus bit *n* contains the value from bit *n* of COLOR0.
B9	Color 1	*Pixel value corresponding to "color 1".* COLOR1 contains the source foreground color to be used during a bit-expand, fill, or draw-and-advance operation. The pixel value should be replicated throughout the 16 LSBs of register B9 (see note below). Nonreplicated patterns may be entered for dithering effects. The 16 MSBs are ignored during the expand operation. For example, at four bits per pixel, COLOR1 contains four identical pixel values, as shown below. 31 28 | 27 24 | 23 20 | 19 16 | 15 12 | 11 8 | 7 4 | 3 0 Pixel | Pixel | Pixel | Pixel | Pixel | Pixel | Pixel | Pixel Each of the 16 LSBs of COLOR1 is associated with the corresponding pin of the local address/data bus, LAD0–LAD15. COLOR1 bit 0 is associated with bit 0 of the data bus (the bit transferred on LAD0), COLOR1 bit 1 is associated with bit 1 of the data bus, and so on. When the contents of COLOR1 are output over a portion of the data bus, including bit *n* of the bus, bus bit *n* contains the value from bit *n* of COLOR1.
B10–B14		*PixBlt temporary registers.* PixBlt instructions use these registers for storing temporary values and context information necessary to resume execution of a partially-completed PixBlt operation in the event of an interrupt.
SP	SP	*Stack pointer.* SP contains the bit address of the top of the stack.

Notes: To provide upward compatibility with future versions of the GSP, replicate the pixel value throughout all 32 bits of COLOR0 or COLOR1, as shown.

B0 SADDR - Source Address Register B0

Syntax B0

Format

31 ... 16	15 ... 0
Y	X

or

31 ... 0
Linear Bit Address

Description

SADDR contains the source array address pointer for PIXBLTs. Generally, SADDR points to the pixel with the lowest address in the source array. When a corner adjust is necessary, the GSP automatically adjusts SADDR to point to the selected starting corner of the source array. (For PIXBLT L,L, however, you must manually adjust SADDR to point to the starting corner. This feature allows you to use PIXBLT L,L for manipulating pixel arrays with pitches that are not powers of two.)

SADDR is in either XY or linear format. If the first operand of a PIXBLT instruction is an ***L*** (such as PIXBLT L,XY), then SADDR is in linear format. If the first operand of a PIXBLT instruction is an ***XY*** (such as PIXBLT XY,L), then SADDR is in XY format.

During PIXBLT operations, SADDR is used in linear format. When the PIXBLT is completed, SADDR points to the starting location of the row that follows the last row in the array. If a PIXBLT is interrupted, SADDR points to the next word of pixels to be read.

During LINE operation, SADDR contains the current decision variable value.

The following instructions use SADDR as an implied operand:

Instruction	SADDR Format and Function
LINE	Contains $d=2b-a$, used for the line draw.
PIXBLT B,L	Linear address; points to the beginning of a binary source array.
PIXBLT B,XY	Linear address; points to the beginning of a binary source array.
PIXBLT L,L	Linear address with special requirements when PBH = 1 or PBV = 1. Refer to the PIXBLT L,L for a description of its unique requirements.
PIXBLT L,XY	Linear address; points to the beginning of a source array.
PIXBLT XY,L	XY address; points to the beginning of a source array.
PIXBLT XY,XY	XY address; points to the beginning of a source array.

Example

```
SADDR  .set   B0
*
       MOVE   >00080015,SADDR    ;Move XY value >15,>8 into
*                                ;B0
       MOVE   >00010AFC,SADDR    ;Move linear value >10AFC
*                                ;into B0
```

Format

31	0
Linear Bit Address	

Description

SPTCH specifies the linear difference in the start addresses of adjacent lines of the source array for PIXBLT and FILL instructions. The GSP uses the value in SPTCH to move from row to row through the source array. SPTCH **must** be an integer multiple of 16 (except for the special cases of PIXBLT B,L and PIXBLT B,XY). SPTCH is constrained in some cases to be a power of two; this allows XY addressing and automatic corner adjust operations.

Some PIXBLTs store an adjusted value of SPTCH during instruction execution. This mechanism is transparent unless the PIXBLT is interrupted. However, the original contents of SPTCH are restored if the instruction is allowed to complete normally.

The following instructions use SPTCH as an implied operand.

Instruction	SPTCH Format and Function
PIXBLT B,L	Linear; unconstrained otherwise.
PIXBLT B,XY	Linear; power of two for windowing; unconstrained otherwise.
PIXBLT L,L	Unconstrained except as previously noted. SPTCH is not related to CONVSP for this instruction; therefore, it is not constrained to be a power of two.
PIXBLT L,XY	Linear; power of two for windowing and PBV = 1; unconstrained otherwise except as previously noted.
PIXBLT XY,L	Power of two.
PIXBLT XY,XY	Power of two.

Example

```
SPTCH  .set   B1
*
       MOVE   >00001000,SPTCH    ;Power of two for
*                                ;PIXBLT XY,L
       MOVE   >00010AFC,SPTCH    ;Unconstrained value for
*                                ;PIXBLT B,L
```

B2 DADDR - Destination Address Register B2

Format

31 ... 16	15 ... 0
Y	X

or

31 ... 0
Linear Bit Address

Description

DADDR specifies the address of the least significant pixel in the destination array for PIXBLTs. Generally, DADDR points to the pixel with the lowest address in the destination array. When a corner adjust is necessary, the GSP automatically adjusts DADDR to point to the selected starting corner of the destination array. (For PIXBLT L,L, however, you must manually adjust DADDR to point to the starting corner. This feature allows you to use PIXBLT L,L for manipulating pixel arrays with pitches that are not powers of two.)

DADDR is also used in conjunction with DYDX to perform a common rectangle function for some instructions (FILL XY, PIXBLT B<XY, PIXBLT L,XY, and PIXBLT XY,XY, with window option 1). In these cases, DADDR is set to the starting XY address of the common pixel block described by the intersection of the original destination array and the pixel block indicated by WSTART and WEND. No drawing is performed. If there is no common array, the V bit is not set and the value of DADDR is indeterminate.

DADDR is in either XY or linear format. If the second operand of the PIXBLT instruction is an ***L*** (such as PIXBLT XY,L), then DADDR is in linear format. If the second operand of the PIXBLT instruction is an ***XY*** (such as PIXBLT XY,XY), then DADDR is in XY format.

During PIXBLT operation, DADDR is maintained in linear format. When the PIXBLT completes, DADDR points to the linear starting address of the row following the last row in the array. If a PIXBLT is interrupted, DADDR points to the next word of pixels to be read.

For the LINE instruction, DADDR contains the XY address of the next DDA drawing point.

The following instructions use DADDR as an implied operand.

Instruction	DADDR Format and Function
FILL L	Linear; points to the beginning of the destination array.
FILL XY	XY; points to the beginning of the destination array.
LINE	XY; points to the current pixel.
PIXBLT B,L	Linear; points to the beginning of the destination array.
PIXBLT B,XY	XY; points to the beginning of the destination array.
PIXBLT L,L	Linear with special requirements when PBH=1 or PBV=1. Refer to the PIXBLT L,L for a description of its unique requirements.
PIXBLT L,XY	XY; points to the beginning of the destination array.
PIXBLT XY,L	Linear; points to the beginning of the destination array.
PIXBLT XY,XY	XY; points to the beginning of the destination array.

Example

```
DADDR .set   B2
*
       MOVE   >00080015,DADDR    ;Move XY value >15,>8 into
*                                ;B2
       MOVE   >00010AFC,DADDR    ;Move linear value >10AFC
*                                ;into B2
```

B3 DPTCH - Destination Pitch Register B3

Format

31	0
Linear Bit Address	

Description

DPTCH specifies the linear difference in the start addresses of adjacent lines of the destination array for PIXBLT and FILL instructions. The TMS34010 uses the value in DPTCH to move from row to row through the destination array. DPTCH **must** be an integer multiple of 16 (except for FILL L when DX=1). DPTCH is also constrained in some cases to be a power of two to allow XY addressing and automatic corner adjust.

Some PIXBLTs store an adjusted value in DPTCH during instruction execution. This mechanism is transparent, unless the PIXBLT is interrupted. The original contents of DPTCH are restored if the instruction is allowed to complete normally.

The following instructions use DPTCH as an implied operand.

Instruction	DPTCH Format and Function
FILL L	Linear; unconstrained for DX=1.
FILL XY	Linear; power of two.
PIXBLT B,L	Linear; unconstrained except as previously noted.
PIXBLT B,XY	Linear; power of two for windowing; unconstrained otherwise except as noted above.
PIXBLT L,L	Linear; unconstrained except as previously noted. DPTCH is not related to CONVDP for this instruction; therefore, it is not constrained to be a power of two.
PIXBLT L,XY	Linear; power of two.
PIXBLT XY,L	Linear; power of two for PBV = 1; unconstrained otherwise except as previously noted.
PIXBLT XY,XY	Linear; power of two.

Example

```
DPTCH  .set   B3
*
       MOVE   >00001000,DPTCH    ;Power of two for
*                                ;PIXBLT XY,L
       MOVE   >00010AFC,DPTCH    ;Unconstrained value for
*                                ;PIXBLT L,L
```

B4 OFFSET - XY Addressing Offset Register

Format

31	0
Linear Bit Address	

Description

OFFSET contains the linear address of the first pixel in the XY coordinate space for instructions using XY addressing. This corresponds to the linear address of the XY origin (X=0,Y=0). This value is used as the memory base for performing XY to linear address conversions.

OFFSET is always in linear format. It may be placed at any position in the TMS34010 linear address space and should contain a pixel-aligned value for proper XY address conversions, transparency, pixel processing, and plane masking. OFFSET is not modified by instruction execution.

The following instructions use OFFSET as an implied operand.

Instruction	OFFSET Format and Function
CVXYL RS,RD	Linear address of XY origin
DRAV RS,RD	Linear address of XY origin
FILL XY	Linear address of XY origin
LINE	Linear address of XY origin
PIXBLT B,XY	Linear address of XY origin
PIXBLT L,XY	Linear address of XY origin
PIXBLT XY,L	Linear address of XY origin
PIXBLT XY,XY	Linear address of XY origin
PIXT RS,RD.XY	Linear address of XY origin
PIXT RS.XY,RD	Linear address of XY origin
PIXT RS.XY,RD.XY	Linear address of XY origin

Example

```
OFFSET .set  B4
*
       MOVE  >00042000,OFFSET  ;Linear value on pixel
*                              ;boundary
```

WSTART - Window Start Address Register

Format

31 — 16	15 — 0
Window start Y	Window start X

Description

WSTART specifies the XY address of the least significant pixel contained in the rectangular destination clipping window. WSTART is valid for instructions that use an XY destination address and a window option. The least significant pixel is the pixel with the lowest address in the array. For a screen with the ORG bit of the DPYCTL register set to 0, this corresponds to the pixel in the upper left corner of the pixel array.

WSTART may be placed at any position in the positive quadrant of the XY address space. It describes an inclusive pixel; that is, the pixel at the XY location contained in WSTART is included in the window. The value in WSTART is used with WEND, DADDR, and DYDX to preclip pixels, lines, and pixel arrays. WSTART is not modified by instruction execution.

The following instructions use WSTART as an implied operand.

Instruction	WSTART Format and Function
CPW RS,RD	XY value of least significant window corner
DRAV RS,RD	XY value of least significant window corner
FILL XY	XY value of least significant window corner
LINE	XY value of least significant window corner
PIXBLT B,XY	XY value of least significant window corner
PIXBLT L,XY	XY value of least significant window corner
PIXBLT XY,XY	XY value of least significant window corner
PIXT RS,RD.XY	XY value of least significant window corner
PIXT RS.XY,RD.XY	XY value of least significant window corner

Example

```
WSTART .set  B5
*
         MOVE   >00400100,WSTART  ;XY value (256,64) stored
*                                 ;in WSTART
```

B6 WEND - Window End Address Register B6

Format

31 … 16	15 … 0
Window end Y	Window end X

Description

WEND specifies the XY address of the most significant pixel contained in the rectangular destination clipping window. WEND is valid for instructions that use an XY destination address and a window option. The most significant pixel is the pixel with the highest address in the array. For a screen with the ORG bit of the DPYCTL register set to 0, this corresponds to the pixel in the lower right corner of the pixel array.

WEND may be placed at any position in the positive quadrant of the XY address space. It describes an inclusive pixel; that is, the pixel at the XY location contained in WEND is included in the window. The value in WEND is used with WSTART, DADDR, and DYDX to preclip pixels, lines, and pixel arrays. WEND is not modified by instruction execution.

The following instructions use WEND as an implied operand.

Instruction	WEND Format and Function
CPW RS,RD	XY value of most significant window corner
DRAV RS,RD	XY value of most significant window corner
FILL XY	XY value of most significant window corner
LINE	XY value of most significant window corner
PIXBLT B,XY	XY value of most significant window corner
PIXBLT L,XY	XY value of most significant window corner
PIXBLT XY,XY	XY value of most significant window corner
PIXT RS,RD.XY	XY value of most significant window corner
PIXT RS.XY,RD.XY	XY value of most significant window corner

Example

```
WEND   .set   B6
*
       MOVE   >00400100,WEND     ;XY value (256,64) stored
*                                ;in WEND
```

Format

31 — 16	15 — 0
Delta Y	Delta X

Description

DYDX specifies the X and Y dimensions of the rectangular destination array for PIXBLT and FILL instructions. Both the X and Y dimensions are in pixels; that is, the DX value is number of pixels in width of the array, and DY is the number of lines of pixels in the destination array.

When the window clipping option is selected, the pixel block dimensions for the transfer are determined by the relationships between WSTART, WEND, DADDR, and DYDX. If either the X or Y dimension is 0, then the block is interpreted as having a dimension of 0; no transfer is performed.

The values for DY and DX may range up to the coordinate extent of the display (up to 65,535, depending on the display pitch and pixel size selected). For window operations, the relationship between DYDX, WSTART, and WEND is such that DY = $WEND_y$ - $WSTART_y$ + 1 and DX = $WEND_x$ - $WSTART_x$ + 1. The value in DYDX is used with WSTART, DADDR, and DYDX to preclip pixels, lines, and pixel arrays.

Most instructions do not modify the contents of DYDX. For FILL XY, PIXBLT B,XY, PIXBLT L,XY, and PIXBLT XY,XY, with window option 1, however, DYDX is used with DADDR to perform a common rectangle function. In this case, DYDX is set to the dimensions of the common pixel block described by the intersection of the original destination array and the window identified by WSTART and WEND. No drawing is performed. If there is no common rectangle, the V bit is not set and the value of DYDX is indeterminate. See these instructions for further information.

The following instructions use DYDX as an implied operand.

Instruction	DYDX Format and Function
FILL L	Array dimensions in XY format.
FILL XY	Array dimensions in XY format; special requirements when W=1 is selected, as previously noted.
LINE	Dimensions of the rectangle described by the line to be drawn.
PIXBLT B,L	Array dimensions in XY format
PIXBLT B,XY	Array dimensions in XY format; special requirements when pick is selected, as previously noted.
PIXBLT L,L	Array dimensions in XY format.
PIXBLT L,XY	Array dimensions in XY format; special requirements when pick is selected, as previously noted.
PIXBLT XY,L	Array dimensions in XY format.
PIXBLT XY,XY	Array dimensions in XY format; special requirements when pick is selected, as previously noted.

Example This example illustrates the relationship of DYDX to WSTART *and* WEND.

```
WSTART .set  B5
WEND   .set  B6
DYDX   .set  B7
*
       MOVE  WEND,DYDX            ;Put WEND into DYDX
       SUBXY WSTART,DYDX          ;Generate (WEND - WSTART)
       ADDI  >10001,DYDX          ;Increment by 1 in each
                                  ;dimension
```

Format

31 28	27 24	23 20	19 16	15 12	11 8	7 4	3 0
Pixel	Pixel	Pixel	Pixel	Pixel	Pixel	Pixel	Pixel

Description

COLOR0 specifies the replacement color for 0 bits in the source array for PIXBLT B,L and PIXBLT B,XY instructions. These two instructions transform binary pixel array information to multiple bits per pixel arrays using the color information in COLOR1 and COLOR0. The lower 16 bits of COLOR0 are used for the 0 or background color. There is a direct correspondence between the alignment of pixels within the COLOR0 register and pixels within memory words to be altered. That is, individual pixels within COLOR0 are used as they align with destination pixels in the destination word.

COLOR0 is not modified by instruction execution.

> **Note:**
>
> The example format above is for four bits per pixel.

The following instructions use COLOR0 as an implied operand.

Instruction	COLOR0 Contents
PIXBLT B,L	Background pixel color for expanded array
PIXBLT B,XY	Background pixel color for expanded array

Example

```
COLOR0 .set  B8
*
       MOVI  >00005555,COLOR0  ;store uniform pixel value
                               ;in COLOR0
```

COLOR1 - Foreground Color Register

Format

31 28	27 24	23 20	19 16	15 12	11 8	7 4	3 0
Pixel	Pixel	Pixel	Pixel	Pixel	Pixel	Pixel	Pixel

Description

COLOR1 specifies the replacement color for pixels to be altered at the destination pixel or pixel block for FILL, DRAV and LINE instructions.

For PIXBLT B,L and PIXBLT B,XY instructions, COLOR1 specifies the replacement color for 1 bits in the source array. These two instructions transform binary pixel array information to multiple-plane pixel arrays using color information in COLOR1 and COLOR0. There is a direct correspondence between the alignment of pixels within the COLOR1 register and pixels within memory words to be altered. That is, individual pixels within COLOR1 are used as they align with destination pixels in the destination word.

COLOR1 is not modified by instruction execution.

> **Note:**
>
> The example format above is for four bits per pixel.

The following instructions use COLOR1 as an implied operand.

Instruction	COLOR1 Contents
DRAV RS,RD	Pixel color for pixel draw
FILL L	Pixel color for filled array
FILL XY	Pixel color for filled array
LINE	Pixel color for line draw
PIXBLT B,L	Foreground pixel color for expanded array
PIXBLT B,XY	Foreground pixel color for expanded array

Example

```
COLOR1.set   B9
*
        MOVI   >00003333,COLOR1  ;Store uniform pixel value
                                 ;in COLOR1
```

Format

31	0
Various Formats	

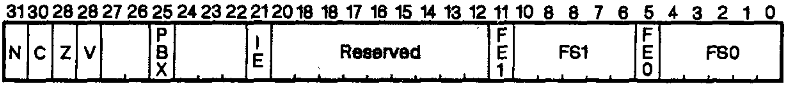
Description

B10 – B14 are used as implied operands for the LINE instruction and as temporary registers for PIXBLTs and FILLs. B13 (PATTRN register) is reserved for future LINE draw enhancement. It should be set to >FFFFFFFF before executing the LINE instruction to ensure software compatibility.

5.2 Status Register

The status register (ST) is a special-purpose, 32-bit register that specifies the processor status. The ST also contains several parameters that specify the characteristics of two programmable data types, fields 0 and 1. The ST is initialized to >00000010 at reset.

Figure 5-4 illustrates the status register. Table 5-2 lists the functions associated with the status bits. Table 5-3 describes the encoding of the field size bits in FS0 and FS1.



Note: The status register bits marked *reserved* (bits 12–20, 22–24, and 26–27) are currently unused. When read, a reserved bit returns the last value written to it. At reset, all reserved bits are forced to 0s.

Figure 5-4. Status Register

Table 5-2. Definition of Bits in Status Register

Bit No.	Field Name	Function
0–4	FS0	*Field Size 0.* Length in bits of first memory data field (see Table 5-3 for values).
5	FE0	*Field Extend 0.* Bit determines whether field from memory is extended with 0s or with the sign bit when loaded into 32-bit general-purpose register. FE0 = 0 – Zero extension FE0 = 1 – Sign extension
6–10	FS1	*Field Size 1.* Length in bits of second memory data field (see Table 5-3 for values).
11	FE1	*Field Extend 1.* Bit determines whether field from memory is extended with 0s or with the sign bit when loaded into 32-bit general-purpose register. FE1 = 0 – Zero extension FE1 = 1 – Sign extension
12–20	–	*Reserved*
21	IE	*Interrupt Enable.* Master interrupt enable/disable bit. IE = 0 – All maskable interrupts disabled IE = 1 – All maskable interrupts enabled
22–24	–	*Reserved*

Table 5-2. Definition of Bits in Status Register (Concluded)

Bit No.	Field Name	Function
25	PBX	*PixBlt Executing.* Indicates upon return from an interrupt that the interrupt occurred between instructions or in the middle of a PIXBLT or FILL instruction. 0 = Indicates interrupt occurred at PIXBLT or FILL instruction boundary 1 = Indicates interrupt occurred in the middle of a PIXBLT or FILL instruction
26–27	–	*Reserved*
28	V	*Overflow.* Set according to instruction execution.
29	Z	*Zero.* Set according to instruction execution.
30	C	*Carry.* Set according to instruction execution.
31	N	*Negative.* Set according to instruction execution.

Table 5-3. Decoding of Field-Size Bits in Status Register

Five FS Bits	Field Size†	Five FS Bits	Field Size†	Five FS Bits	Field Size†	Five FS Bits	Field Size†
00001	1	01001	9	10001	17	11001	25
00010	2	01010	10	10010	18	11010	26
00011	3	01011	11	10011	19	11011	27
00100	4	01100	12	10100	20	11100	28
00101	5	01101	13	10101	21	11101	29
00110	6	01110	14	10110	22	11110	30
00111	7	01111	15	10111	23	11111	31
01000	8	10000	16	11000	24	**00000**	**32**

† In bits

5.3 Program Counter

The program counter (PC) is a dedicated 32-bit register that points to the next instruction word to be executed. Instructions are always aligned on even 16-bit word boundaries, and as shown in Figure 5-5, the four LSBs of the PC are always 0s.

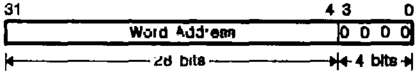


Figure 5-5. Program Counter

An instruction consists of one or more instruction words. The first word contains the opcode for the instruction. Additional words may be required for immediate data or absolute addresses. As each instruction word is fetched, the PC is incremented by 16 to point to the next instruction word. The PC contents are replaced during a branch instruction, subroutine call instruction, return instruction, or interrupt. Instructions may be categorized according to their effect on the PC, as indicated in Table 5-4.

Table 5-4. Instruction Effects on the PC

Category	Description
Non-branch	The PC is incremented by 16 at the end of the instruction, allowing execution to proceed sequentially to the next instruction.
Absolute Branch (TRAP, CALL, JAcc)	The PC is loaded with an absolute address; the four LSBs of the address are set to 0s.
Relative Branch (JRcc, DSJxx)	The signed displacement (8 or 16 bits) is added to the current contents of the PC. The signed displacement is treated as a word displacement; that is, it is shifted left four bit positions before it is added to the PC.
Indirect Branch (JUMP, CALL, EXCPC)	The PC is loaded with the register contents.The four LSBs are set to 0s.

5.4 Instruction Cache

Most program execution time is spent on repeated execution of a few main procedures or loops. Program execution can be speeded up by placing these often used code segments in a fast memory. The TMS34010 uses a 256-byte instruction cache for this purpose.

Only memory words that are pointed to by the PC can be accessed from the cache. This includes opcodes, immediate operands, and absolute addresses. Instructions and data may reside in the same area of memory; therefore, data could be copied into the instruction cache. However, the processor cannot access data from the cache. All reads and writes of data in memory bypass the cache.

5.4.1 Cache Hardware

The instruction cache contains 256 bytes of RAM, used to store up to 128 16-bit instruction words. Each instruction word in cache is aligned on an even word boundary. Figure 5-6 illustrates cache organization.

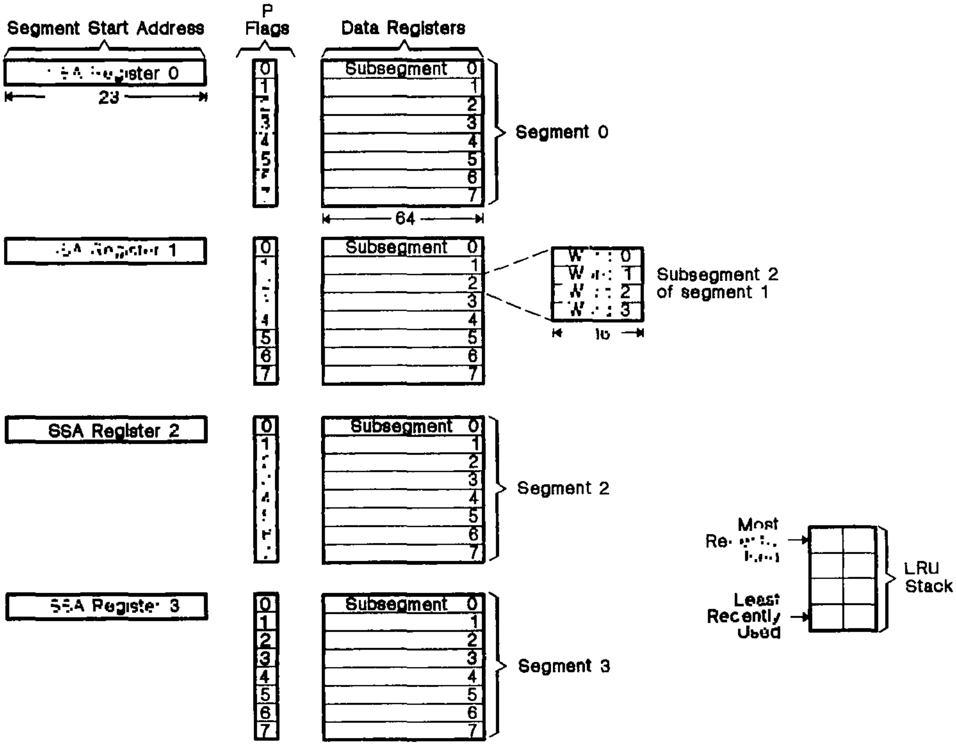


Figure 5-6. TMS34010 Instruction Cache

The cache is divided into four 32-word segments. Each cache segment may contain up to 32 words of a 32-word segment in memory. This memory segment is a block of 32 contiguous words beginning at an even 32-word boundary in memory.

Each cache segment is divided into eight subsegments; each subsegment contains four words. Dividing each segment into subsegments reduces the number of word fetches required from memory when fewer than 32 words of a memory segment are used. Each of the four cache segments is associated with a segment start address (SSA) register. Figure 5-7 shows how an instruction word is partitioned into the components used by the cache control algorithm.

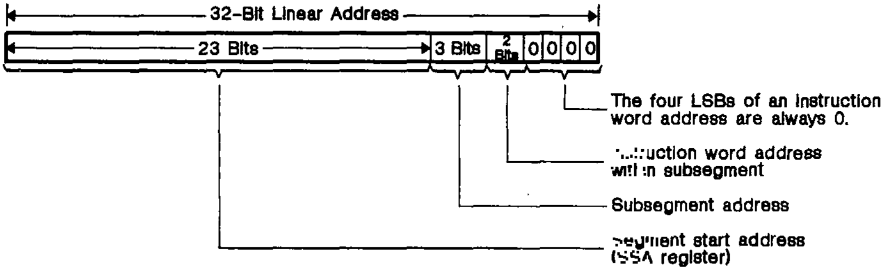


Figure 5-7. Segment Start Address

The 23 bits of the SSA register correspond to the 23 MSBs of the segment's memory address. These 23 MSBs are common to all eight subsegments within a segment. The next three bits (bits 6–8) identify one of the eight subsegments. Bits 4 and 5 identify one of the four words contained in a subsegment. The four LSBs are always 0s because instructions are aligned on word boundaries.

5.4.2 Cache Replacement Algorithm

When the TMS34010 requests an instruction word from a segment that is not in the cache, the contents of one of the four cache-resident segments must be discarded to make room for the segment that contains the requested word. A modified form of the least-recently-used (LRU) replacement algorithm is used to select the segment to be discarded.

The LRU segment manager (an element of the cache control logic) maintains an LRU stack to track use of the four segments. The LRU stack contains a queue of segment numbers, 0 through 3. Each time a segment is accessed, its segment number is placed on the top of the stack, pushing the other three segment numbers down by one position. Thus, the number at the top of the LRU stack identifies the most-recently-used segment and the number at the bottom identifies the least-recently-used segment.

When a new segment must be loaded into cache, the least-recently-used segment is discarded. The eight P flags (described in Section 5.4.3) of the selected segment are set to 0s, and the segment's SSA register is loaded with the new segment address. After the requested subsegment has been loaded from memory, its P flag is set to 1, and the requested instruction fetch is allowed to complete.

Following a reset, all P flags in the cache are set to 0 and the four segment numbers in the LRU stack are stored in numerical order (0–3).

5.4.3 Cache Operation

When the TMS34010 requests an instruction word, it checks to see if the word is contained in cache. First, it compares the 23 MSBs of the instruction address to the four SSA registers. If a match is found, the processor searches for the appropriate subsegment. A present (P) flag, associated with each subsegment, indicates the presence of a particular subsegment within a cache segment. P=1 indicates that the requested word is in cache; this is called a cache hit. If there is no match, or if there is a match and P=0, the word is not in cache; this is called a cache miss.

- **Cache Hit**

  The cache contains the requested instruction word. The processor performs the following actions:

  1) A short access cycle reads the instruction word from cache.
  2) The segment number is moved to the top of the LRU stack, pushing the other three segment numbers toward the bottom of the stack.

- Cache Miss

  The cache does not contain the instruction word. There are two types of cache miss - subsegment miss and segment miss.

  Subsegment Miss. The 23 MSBs of the instruction word address match one of the four SSA registers' 23 MSBs; that is, the appropriate segment is in the cache. However, the P flag for the requested subsegment is not set. The processor performs the following actions:

  1) The four-word subsegment containing the requested instruction word is read from local memory into the cache.
  2) The segment number is moved to the top of the LRU stack, pushing the other three segment numbers toward the bottom of the stack.
  3) The subsegment's P flag is set.
  4) The instruction word is read from the cache.

  Segment Miss. The instruction word address does not match any of the SSA registers. The processor performs the following actions:

  1) The least-recently-used segment is selected for replacement; the P flags of all eight subsegments are cleared.
  2) The SSA register for the selected segment is loaded with the 23 MSBs of the address of the requested instruction word.
  3) The four-word subsegment in memory that contains the requested instruction word is read into the cache. It is placed in the appropriate subsegment of the least-recently-used segment. The subsegment's P flag is set to 1.
  4) The LRU stack is adjusted by moving the number of the new segment from the bottom (indicating that it is least recently used) to the top (indicating that it is most recently used). This pushes the other three segment numbers in the stack down one position.
  5) The instruction word is read from the cache.

5.4.4 Self-Modifying Code

Avoid using self-modifying code; it can cause unpredictable results. When a program modifies its own instructions, only the copy of the instruction that resides in external memory is affected. Copies of the instructions that reside in cache are not modified, and the internal control logic does not attempt to detect this situation.

5.4.5 Flushing the Cache

Flushing the cache sets it to an initial state which is identical to the state of the cache following reset. The cache is empty and all 32 P flags are set to 0.

The cache is flushed by setting the CF (cache flush) bit in the HSTCTL register to 1. The CF bit retains the last value loaded until a new value is loaded or until the GSP is reset. The contents of the cache remain flushed as long as the CF bit is set to 1. All instruction fetches bypass the cache and are accessed directly from memory.

Unless the cache is disabled, normal cache operation will resume when the CF bit is set to 0.

One use for flushing the cache is to facilitate downloading new code from a host processor to GSP local memory. The host typically halts the GSP during downloading by writing a 1 to the HLT bit in the HSTCTL register. Before allowing the GSP to execute downloaded code, the host should flush the cache as described in Section 5.4.5.

5.4.6 Cache Disable

Disabling the cache facilitates program debugging and emulation. The cache is disabled by setting the CD (cache disable) bit in the CONTROL register to 1. While disabled, the cache is bypassed and all instructions are fetched from external memory.

CD=1 has the same effect as CF=1 with one exception. While CD=1 and CF=0, data already in the cache are protected from change. When the CD bit is set back to 0, the state of the cache prior to setting the CD bit to 1 is restored. The instructions in the cache are once again available for execution. If the contents of the cache become invalid while CD=1, they can be flushed by setting CF to 1.

The CD bit can be manipulated to preserve code in the cache for faster execution in some time-critical applications. For example, if an inner loop just exceeds 256 bytes, most of the loop, but not all of it, can fit in the cache. During execution of the few instructions that are not in the cache, the CD bit can be set to 1 to prevent the code in the cache from being replaced. In this instance, the loop's execution speed is improved by eliminating the thrashing of cache contents. Use this technique carefully; in some cases, it can negatively affect execution speed.

5.4.7 Performance with Cache Enabled versus Cache Disabled

When the instruction cache is disabled, instruction words are fetched from external memory. Assuming no wait states are necessary, each instruction fetch from external memory adds 3 machine cycles to the access time. This is considerably slower than a program which uses the cache efficiently (when each word in cache is used several times before it is replaced).

An inefficient use of cache occurs when words in cache are used only once before replacement. This produces a cache miss every fourth word. With the cache enabled, the time penalty due to cache misses in this case is 2.25 machine states per instruction, calculated as follows:

- Eight machine cycles are required to load four words into cache from memory
- An additional machine state is required to process the instruction
- Dividing the total of nine machine states by four instructions yields an average of 2.25 machine states per instruction

Performance using the cache is nearly always better than performance with the cache disabled. The only exception occurs when the code contains so many jumps that only a portion of each subsegment is executed before control is transferred to another subsegment.

5.5 Internal Parallelism

Figure 5-8 illustrates the internal data paths associated with TMS34010 processor functions. The TMS34010 has a single, logical memory space for storage of both data and instructions. However, internal parallelism provides the GSP with the benefits found in architectures which contain separate data and instruction storage. The ability to fetch instructions from cache in parallel with data accesses from memory greatly enhances execution speed. Hardware parallelism allows the following three storage areas to be accessed simultaneously:

- Instruction cache
- Dual-ported, general-purpose register files A and B
- External memory

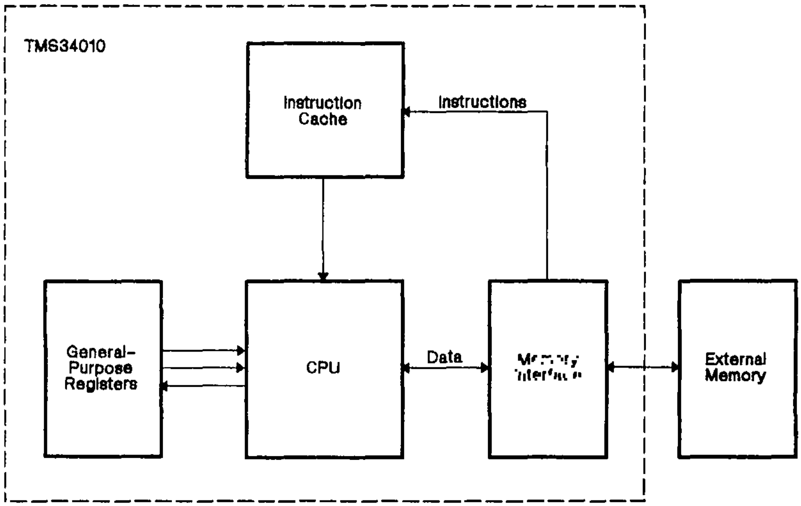


Figure 5-8. Internal Data Paths

Each storage area can also be accessed independently of the other two. This allows the GSP to perform the following actions in parallel during each pair of machine states:

- One external memory cycle
- Two instruction fetches from cache
- Four reads and two writes to the general-purpose register files

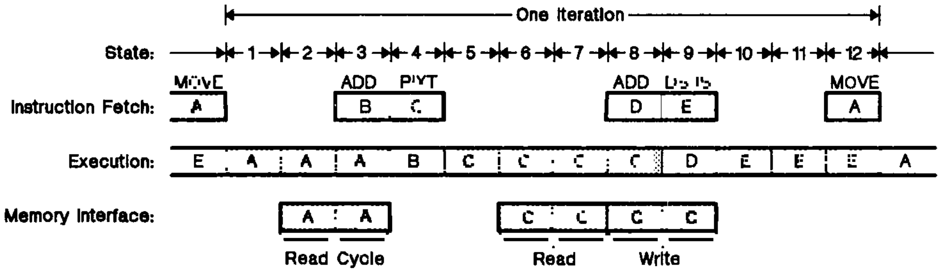
The need to schedule conflicting internal operations can limit the GSP's ability to perform these actions in parallel. For example, an instruction which requires the memory controller to perform a read must complete before the next instruction can be executed.

Figure 5-9 illustrates an example of internal parallelism. Figure 5-9 *a* shows three activities occurring in parallel:

- Instructions are fetched from cache.
- Instructions are executed through the general-purpose registers and the ALU.
- The local memory interface controller performs memory accesses.

Figure 5-9 *a* represents execution of the code in Figure 5-9 *b*, which is the inner loop of a graphics routine. The memory controller accesses pixels while the ALU fetches instructions from cache. The memory controller completes a write cycle while execution begins on the next instruction.

(a)



(b)

```
A    L1:   MOVE   *B1+,B10,0   Get DELTAX
B          ADD    B10,B8       Adjust pixel pointer
C          PIXT   *B1,*B8      Draw next pixel
D          ADD    B0,B1        Add field size
E          DSJS   B11,L1       Loop N Times
```

Figure 5-9. Parallel Operation of Cache, Execution Unit, and Memory Interface